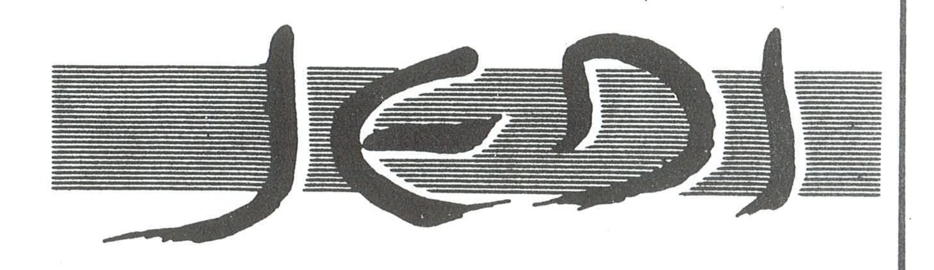


19

QUE LE FORTH SOIT AVEC VOUS

DECEMBRE 1985

FORTH
PILOT
MUMPS
COBOL
FUTURLOG

pour : ORIC
AMSTRAD
APPLE
JUPITER ACE
HECTOR HRX
ATARI
FUTURSYS

EDITORIAL

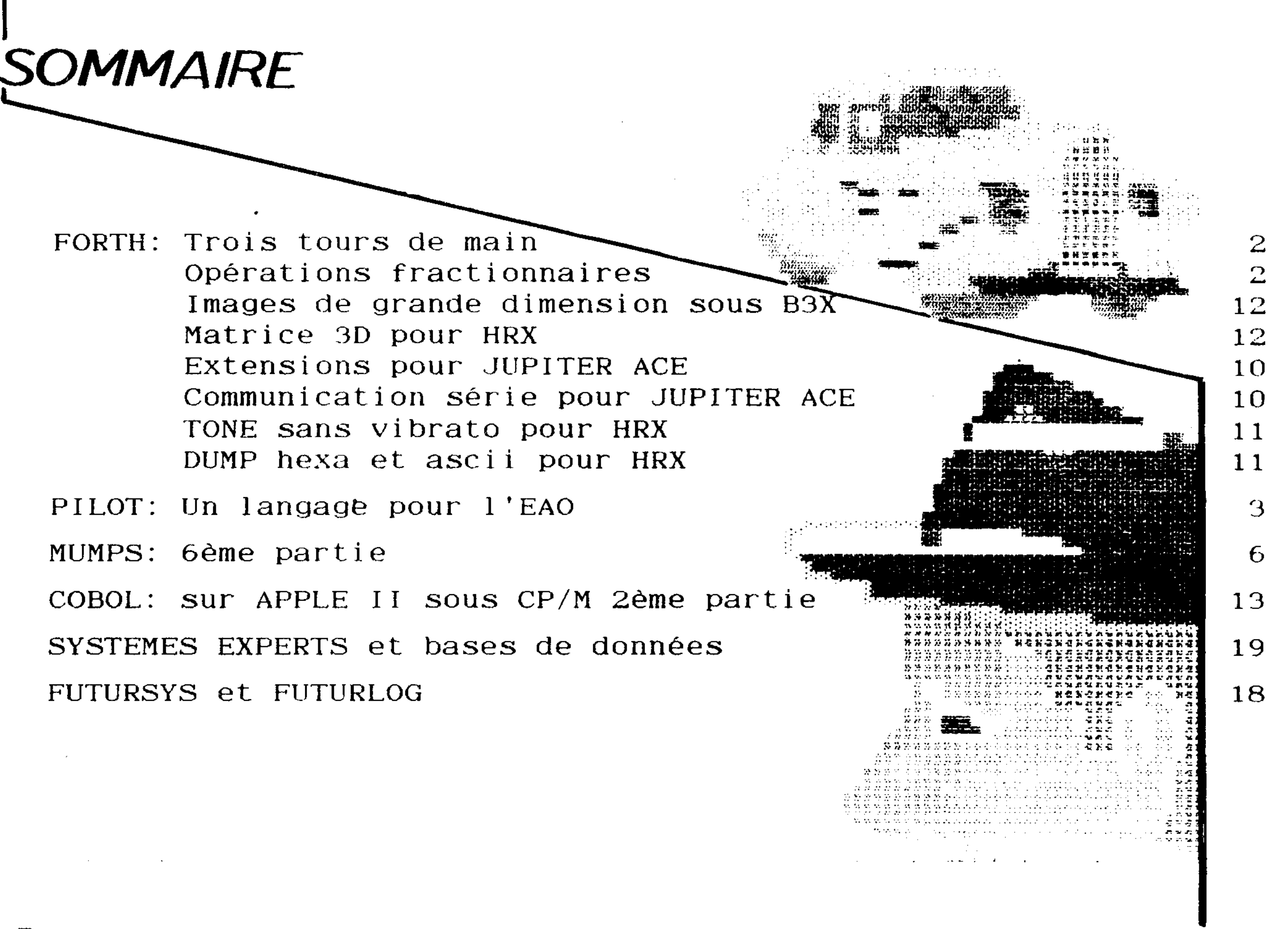
Ce mois-ci, JEDI marque son entrée en FORTH dans le monde de la télématique. En effet, c'est depuis peu, et malgré quelque retard par rapport à la date prévue, que nous proposons sur le serveur SAM des programmes téléchargeables. Donc, d'ici peu, tout programme pour TO7 et TO7/70 sortant de nos ateliers sera expliqué et listé dans la revue, mais également diffusé par SAM. Ce nouveau service illustre bien l'intérêt des relations possibles et fructueuses entre associations et sociétés publiques ou privées. Déjà dans un passé récent, JEDI et MICRO KIT s'associaient à l'occasion du SICOB 85 pour monter un serveur sur le stand JEDI. Les retombées de cette collaboration permettent maintenant d'apporter aux utilisateurs du VEGAS 6809 une assistance technique accessible par MINITEL.

Une autre orientation de JEDI concerne les systèmes experts et l'intelligence artificielle. Très critiqué, le terme IA suggère trop souvent une image du programmeur similaire à celle du docteur Frankenstein. Or l'IA, au sens large, et ce qui en découle, sera, nous n'en doutons point, le moteur de l'informatique et de la télématique des prochaines années. Réfléchissons un peu là dessus et voyons les possibilités; il est des domaines, le diagnostic de pannes automobiles ou l'expertise fiscale pour exemple, demandant des investissement considérables en temps et en argent pour mettre en place une application. Ces systèmes seront d'un prix inabordables pour un usager désirant une expertise ponctuelle. En effet, qui investirait 4000, 10 000 ou 50 000 Fr dans un abonnement mensuel ou annuel à une base de données d'expertise spécifique ou dans une série de disques offrant le même service, avec le risque de sous-employer ce produit.

Dans ce domaine, la concurrence jouera aussi, et un service d'expertise par télématique respectant le dicton "moins cher pour plus de monde" aura beaucoup plus de succès que les banques de données actuelles. Les systèmes experts peuvent s'appliquer à tous les domaines: linguistique, médecine, électronique, mécanique, fiscalité, marketing, etc...

Encore faut-il maitriser cette technique nouvelle, choisir le bon langage et le bon système, évaluer l'objectif de l'expertise et connaitre ses limites. Alors suivez-nous et aidez-nous. Vos suggestions, articles et travaux sont les nôtres.

SOMMAIRE



ASSOCIATION JEDI 8, rue Poirier de Narçay 75014 PARIS
Tel: (1) 542.88.90 (de 10h à 18h)
(1) 45.42.88.90 à partir du 25Oct1985

FORTH Trois petits tours de main

par J.Y. SCLAVON

```
SCR # 1
 ( CODAGE A CLEF UNIQUE )
 ( TOUTES MACHINES        )

 : CODE ( C AD NB -- )
 Ø DO
      2DUP I + DUP ROT
      SWAP @ XOR SWAP !
 LOOP
 2DROP ;

 ( C  : CODE SUR 16 BITS          )
 ( AD : ADRESSE DE DEPART         )
 ( NB : NB D'OCTETS A CODER       )

SCR # 2
 ( UTILITAIRES HIRES POUR ORIC)

 : THIRES ( -- )
 HIRES TEXT
 46000 336 255 FILL
 46352  32   Ø FILL
     3 704     C! ;

 : | ( -- )
 48000 C@ Ø= IF 31 ELSE Ø THEN
 48000 C! ;

 ( THIRES : INITIALISATION        )
 (      | : BASCULE HIRES/TEXT )

 ;S

SCR # 3
 ( COPIE HIRES POUR ORIC -> MT-8Ø )
 : J R> R> R> R> DUP >R
 SWAP >R SWAP >R SWAP >R ;

 : HCOPY PON
 27 EMIT 51 EMIT 16 EMIT
 39 Ø DO 13 EMIT 1Ø EMIT
 27 EMIT 75 EMIT 2ØØ EMIT Ø EMIT
 92Ø 892Ø DO 4ØØØØ I J + +
 C@ 64 - DUP Ø< IF DROP THEN
 EMIT -4Ø +LOOP LOOP
 CR 27 EMIT 64 EMIT CR
 POF ;

 ;S
```

Ecran 1 :

Codage et décodage d'une zone mémoire à l'aide d'un nombre 16 bits.

ATTENTION : Programme vicieux ! En effet, toute tentative de décodage avec un autre nombre que celui qui a servi au codage provoque un codage suplémentaire (la probabilité de "retomber" sur l'original après plusieurs essais est très faible).

Ecran 2 :

THIRES est le mot d'initialisation qui permet de visualiser (au besoin de modifier) le haut de la page HIRES tout en conservant la page TEXT. Mais en raison de la taille de la page TEXT, la haute résolution se limite alors à 240*120 point, ce qui s'avérera suffisant pour créer des effets surprenans (icones par exemple ...) pour une machine de cette envergure.

\ fonctionne en bascule et inverse l'état de l'écran à chaque fois qu'il est invoqué.

Ecran 3 :

HCOPY recopie fidèlement la page HIRES sur la MT-80 (les cercles sont circulaires, compas à l'appui !).

ORIC 1

FORTH Opérations fractionnaires

par J.L. BECHENNEC

Le langage FORTH ne disposant pas en standard des fonctions arithmétiques en virgule flottante, il peut être néanmoins intéressant de manipuler des nombres sous une forme fractionnaire. Et puis, en y réfléchissant bien 1/3 est toujours plus précis que 0,33333333...

Les différentes routines décrites ci-après ont été développées à partir du FORTH AMSTRAD (celui-là même que nous avons implanté) et permettent les opérations suivantes:

PGCD　　n1 n2 --- pgcd

Délivre un nombre n tel que celui-ci soit le PGCD de n1 et n2.

PPCM　　n1 n2 --- ppcm

Délivre un nombre n tel que celui-ci soit le PPCM de n1 et n2.

FRINV　　n1 n2 --- n2 n1

Délivre sur la pile la fraction inverse représentée par n1 et n2. Exemple: soit 3/7, son inverse est obtenu par 3 7 FRINV, ce qui laisse sur la pile les nombres 7 3.

REDUIT　　nn nd --- nr' nd'

Délivre sur la pile deux valeurs correspondant au numérateur et au dénominateur d'une fraction simplifiée. Exemples:

```
5 15 REDUIT . . affiche 1 3 ( soit 1/3)
7 11 REDUIT . . affiche 7 11
```

FR*　　fr1 fr2 --- fr1*fr2
pour fr correspondant à nn nd

Effectue le produit de deux fractions. Exemple:

```
1 3 6 15 FR* . . affiche 6 45
( soit (1/3)*(6/15))
6 45 REDUIT . . affiche 1 5
```

FR/　　fr1 fr2 --- fr1/fr2

Effectue la division de deux fractions. Mêmes principes que FR*.

FR+　　fr1 fr2 --- fr1+fr2

Effectue la somme de deux fractions. idem FR*.

SUITE PAGE 9 →

PILOT : UN LANGAGE POUR L'EAO

Depuis quelque temps, il n'est plus question que d'EAO et de plan IPT. Pour ce faire on propose une foule de "méthodes" aux noms plus ou moins "fruités" et dont la simplicité de mise en oeuvre (meme quand il s'agit de systèmes auteurs) n'égale guère le "tire la bobinette et la chevillette cherra" du conte. Or, il faut se rendre à l'évidence : la société basculant rapidement dans un système de communications sophistiquées et subissant de ce fait une accélération des échanges d'informations, il devient absolument nécessaire de produire des didacticiels performants, capables de seconder efficacement les professeurs. Voilà pour le couplet économique. Fermons le ban et passons aux choses sérieuses.

A dire vrai, de nombreux langages permettent la réalisation de logiciels suffisamment interactifs pour supporter le qualificatif d'éducatifs. Parmi ceux-ci, il en est un, peu connu en Europe, mais qui, a notre sens, mérite largement qu'on le réhabilite. Nous voulons parler de PILOT. Créé au début des années 60 par John Starkweather du California Medical Center de San Francisco, il trouve sa maturité dans les travaux du Dr. Dean Brown du Laboratoire de Propédeutique du Standford Institute, travaux s'échelonnant de 1967 à 1974, et qui prouvèrent la parfaite adéquation du langage à l'enseignement assisté par ordinateur ainsi qu'à l'apprentissage de la programmation par les enfants. PILOT signifie Programmed Inquiry, Learning Or Teaching.

Généralement écrit en PASCAL, ce langage présente de grands avantages sur le BASIC. Tout d'abord, destiné à manipuler des questions-réponses, il est doté de fonctions de comparaison de chaines bien plus simples que celles du BASIC et de la plupart des autres langages. Cela ne veut d'ailleurs pas dire que ces fonctions sont simplistes, loin de là !

Ensuite, bien que doté de numéros de ligne, il fonctionne la plupart du temps en employant des sous-programmes labellisés, ce qui permet de concevoir des applications modulaires directement réutilisables dans d'autres programmes.

Enfin, il intègre souvent une tortue graphique, ce qui permet d'apprendre les primitives des versions enfantines de LOGO.

LE LANGAGE TEL QU'EN LUI MEME

Ainsi que nous vous l'avons déjà dit, PILOT est surtout destiné à des applications textuelles. Aussi, la première instruction est-elle, bien évidemment, conçue pour permettre d'imprimer du texte! Elle s'appelle T:
Si vous tapez
T: BONJOUR !
BONJOUR !
s'inscrira immédiatement à l'écran, ceci parce que vous etes en mode immédiat. Pour passer en mode programme, il suffit d'attribuer un numéro de ligne à votre instruction. Ainsi
10T: BONJOUR !
ne sera suivi d'aucune exécution tant que vous ne ferez point un RUN. Déjà à ce stade il convient de faire trois petites remarques:
-il est important de ne pas séparer le numéro de ligne de la

→

commande T:
- si vous désirez présenter votre programme de façon plus lisible, il sera nécessaire d'utiliser des indentations. Celles-ci sont possibles dès que l'on place une virgule derrière le numéro de ligne. Ainsi :

```
10,  T: BONJOUR
20T:LE FOND DE L'AIR EST FRAIS.
```

Il est tout à fait possible d'enchainer plusieurs chaines de caractères. Pour ce faire, vous ferez suivre la première chaine d'un "backslash". Ainsi :

```
10T: BONJOUR !
20T: LE FOND DE L'AIR EST FRAIS
```

donnera lors de son exécution

```
BONJOUR ! LE FOND DE L'AIR EST FRAIS
```

Vous pourrez, bien entendu, LISTer votre programme et insérer de nouvelles lignes, en supprimer, voire meme RENuméroter votre application; ce qui aura pour effet de multiplier par dix les numéros de ligne. Ainsi :

```
1T: BONJOUR
2T: LE CIEL EST BLEU
REN
```

donnera lors du LISTage

```
10T: BONJOUR
20T: LE CIEL EST BLEU
```

Si vous voulez commenter votre programme (ce à quoi nous vous engageons vivement !), il vous suffira d'utiliser la commande R:

Exemple :

```
10R: ********************************
20R: *                              *
30R: *   DIDACTICIEL HISTOIRE       *
40R: *                              *
50R: ********************************
60T: IL ETAIT UNE FOIS UN PETIT CHAPERON ROUGE
```

donnera après un RUN

```
IL ETAIT UNE FOIS UN PETIT CHAPERON ROUGE
```

Tandis que LIST fera s'afficher

```
10R: ********************************
20R: *                              *
30R: *   DIDACTICIEL HISTOIRE       *
40R: *                              *
50R: ********************************
60T: IL ETAIT UNE FOIS UN PETIT CHAPERON ROUGE
```

A ce propos, nous vous conseillons d'employer la méthode suivante:
Toujours placer les commentaires en début de module et leur attribuer une dimension réduite (10 à 20 lignes maximum)

Mais si la commande T: est indiscutablement très utile, elle ne nous mène pas très loin. Encore faut-il pouvoir rentrer des réponses et les comparer avec des réponses type.

LES COMMANDES A: ET M:

La commande A: est une instruction acceptant votre entrée.
Voici un exemple de son emploi :

→

```
10T: QUELLE EST LA VITESSE DU SON ?
20A:
```

Il est évident qu'il devient nécessaire de donner une réponse type pour pouvoir continuer le QUIZZ. Cette réponse, c'est vous qui la fournirez dans le didacticiel en faisant suivre la demande d'input A: du modèle de réponse qui y sera comparé. Ce modèle s'introduit au moyen de la commande d'unification M: (M pour Match). Exemple :

```
10T: QUELLE EST LA VITESSE DU SON ?
20A:
30M:330 m/s,330 mètres par seconde, trois cent trente mètres/seconde
```

Première constatation : la commande M: admet plusieurs comparatifs. Il est ainsi possible d'obtenir la comparaison avec plusieurs réponses possibles qui peuvent etre soit identiques, soit totalement différentes. Ainsi peut-on permettre à l'étudiant plusieurs solutions à un meme problème quand cela s'avère nécessaire.
Mais il est bien beau d'opérer une comparaison si ce n'est pas pour signaler la justesse (ou la fausseté) de la réponse fournie. Nous retrouvons alors notre commande T:, mais qui s'assortit cette fois d'un Y ou d'un N. L'exemple suivant va vous faire immédiatement comprendre leur emploi.

```
10T: QUELLE EST LA VITESSE DU SON ?
20A:
30M:330 m/s,330 mètres par seconde, trois cent trente mètres/seconde
40TY: TOUT A FAIT JUSTE !
50TN: A COTE DE LA PLAQUE !
```

Si vous avez répondu 300 m/s à la question, vous verrez s'afficher "TOUT A FAIT JUSTE !"
Si, par contre, vous avez confondu avec la vitesse de la lumière, c'est le message "A COTE DE LA PLAQUE !" qui s'affichera.
Vous pourrez d'ailleurs suivre l'exécution séquentielle du programme en tapant TRACE ON. Cela donnera ceci :

```
-- 10T: QUELLE EST LA VITESSE DU SON ?
 QUELLE EST LA VITESSE DU SON ?
-- 20A:
 (votre réponse)
-- 30M:330 m/s,330 mètres par seconde, trois cent trente mètres/seconde
-- 40TY: TOUT A FAIT JUSTE !
40TY: TOUT A FAIT JUSTE !
```

Pour sortir de la fonction TRACE, il vous suffira de faire un TRACE OFF.
Mais ce type de programmation est encore très rudimentaire, du fait de son aspect exclusivement séquentiel. Aussi doit-on etre à meme de permettre à l'étudiant d'essayer plusieurs fois de trouver la bonne réponse, tout en lui fournissant au fur et à mesure des explications complémentaires qui le mettront sur la voie. Comment faire ? C'est ce que nous verrons le mois prochain !

■

VIII NOTION DE LIGNES EN MUMPS

A) Structure des lignes

Tout le code MUMPS est organisé sous forme de lignes. Chaque ligne contient une ou plusieurs instructions (ou commandes) et se termine par un retour chariot (souvent appelé <return> ou <cr>).

La ligne de code MUMPS n'est pas limitée à la ligne physique de l'écran (souvent de 80 caractères). Elle peut, en fait, contenir 255 caractères, l'étiquette y compris. Nous reviendrons ultérieurement sur ce qu'est une etiquette. Le retour chariot ne fait pas partie de la ligne en tant que caractère. MUMPS reconnait deux types de lignes :

- Les lignes de commande

- Les lignes de programme (ou routine)

B) Lignes de commande

Une ligne de commande est constituée de une ou plusieurs instructions (commandes) que vous entrez en vue d'une exécution immediate. C'est ce que nous appelons être en mode direct, par contradiction au mode indirect (appelé souvent mode programme).

MUMPS reconnait qu'une ligne entrée au terminal est une ligne de commande selon son format. La forme générale d'une ligne de commande est la suivante :

```
{commande} {argument(s)}.......{commande} {argument(s)}{ ;commentaire}<cr>
```

Dans laquelle :

- commande représente tous verbes valides (commande) Exemple: Write etc...

- argument(s) correspond au(x) argument(s) de la commande

- commentaire représente une remarque. Nous y reviendrons dans un prochain paragraphe

- <cr> représente le retour chariot.

Exemple :

```
READ !,"entrez votre poids (en Kilos) : ",POIDS SET PT=POIDS/1000 WRITE
"votre poids est de : ",PT," tonne(s)"
```

Dans cet exemple, l'exécution de cette ligne a engendré plusieurs actions. Le curseur est passé à la ligne; le message entrez votre poids (en Kilos) est émis; la machine attend une entrée au clavier, affecte à la variable POIDS la saisie; le contenu de la variable PT est alimenté par le resultat du calcul POIDS/1000; le message votre poids est de : est affiché, suivi du contenu de la variable PT, puis du libellé " tonne(s)".

C) Lignes de programme

Une ligne de programme est constituée d'une ou plusieurs instructions entrées pour une exécution differée/ultérieure. Quand on a validé cette ligne à l'aide du retour chariot, MUMPS la stocke en mémoire. Le code spécial <tab> doit être présent dans une ligne de programme afin que MUMPS fasse la différence entre une ligne de programme et une ligne de commande. Une étiquette (ou label) peut etre insérée en tête d'une ligne de commande. Celle-ci sera séparée du corps de la ligne par le caractère <tab>. Dans le cas ou la ligne n'a pas d'étiquette, elle commence par le caractère <tab>.

Voici la forme générale d'une ligne de programme :

```
{label}<tab>{commande} {arguments}.....{commande} {arguments}{ ;commentaire}
```

→

Ou :

- label est équivalent à une etiquette

- <tab> est la touche spéciale tab

- commande est le verbe (ou instruction/commande)

- arguments est le ou la liste des arguments associés à la commande

- comentaire est la remarque associée à la ligne

Exemple :

DEBUT<tab>W #,"mumps vous dit bonjour",! ;1ere ligne de programme

D) Etiquette

L'étiquette d'une ligne est un nom qui permet d'identifier une ligne. Elle répond, comme pour les variables, à la meme règle de définition. Mais elle ne doit, en aucun cas, dépasser huit caractères. Bien entendu, une étiquette ne doit apparaître qu'une seule fois dans un programme. Le label est particulièrement utile pour les branchements.

E) Commentaire

Le dernier paramètre d'une ligne peut être une remarque (commentaire). Il est isolé du reste de la ligne à l'aide d'un point virgule (;). Et il est constitué de n'importe quel texte composé avec les caractères ASCII visualisables. MUMPS n'exécutera jamais ce qui se trouve derrière un point virgule. Lorsqu'on écrit des programmes, il est tres vivement recommandé d'utiliser les commentaires pour expliquer les traitements. Pensez à la maintenance !!! On peut, également, trouver un commentaire seul sur une ligne de programme, avec ou sans etiquette.

Exemples :

<tab>;ceci est un commentaire

DEBUT<tab>;le programme commence à cette ligne

NOTE : Si un commentaire suit des commandes un espace doit etre inséré entre le(s) dernier(s) argument(s) et le point virgule.

F) Le caractère <tab>

Le caractère <tab> (valeur décimale 9 dans la table ASCII) est l'un des caractères de contrôle non visualisables, disponnibles en MUMPS. Si votre clavier ne comportait pas la touche tab générant le code <tab>, vous pourriez le simuler à l'aide de la touche contrôle (ctrl) et la touche I, entrées simultanément.

IX COMMANDES DE CONDITION SUR/DANS LES LIGNES

A) La commande IF

Une des plus grandes aptitudes des ordinateurs réside dans le fait qu'ils sont capables d'exécuter des ordres de facon conditionnelle. MUMPS permet de spécifier, de plusieurs manières, l'expression d'une condition applicable à une action. Nous avons déja vu, dans les chapitres précédents, la fonction $SELECT ainsi que la mise en oeuvre des postconditions pour les commandes. Maintenant, nous allons étudier la commande IF qui s'applique cette fois-ci, non pas à un verbe mais, à une ligne. La condition mentionnée avec la commande IF implique que le reste de la ligne est, ou non, exécuté (partie à droite de la commande IF). Par exemple, disons que la variable X est égale à 5 et la variable Y est égale à 3. Considérons, maintenant, la ligne de commande suivante :

```
IF Y<X SET Z=Y#X WRITE "Z = ",Z
```

Dans cet exemple, nous voyons que la condition Y inférieur à X est vérifiée. MUMPS, dans ce cas, exécutera les ordres SET et WRITE. La décomposition de la ligne est la suivante :

- test de la condition

- alimentation de la variable Z avec le résultat du calcul Y modulo X

- envoi à l' écran du message Z =   suivi du contenu de la variable Z

Si l'on inversait la condition, à savoir X<Y, aucune action ne se produirait.

A certaines occasions, lorsque nous nous exprimons, la dernière condition que l'on avait évoquée est sous-entendue. MUMPS permet également cette forme de pensée. Analysons la suite de lignes suivante, apres avoir affecté à la variable AGE la valeur 16 et à la variable REGION la chaîne de caractères "Alsace".

```
IF (AGE>15)&(AGE<22)&(REGION="Alsace") W !
IF  W "puisque vous avez ",AGE," ans"
IF  W !,"et que vous habitez en ",REGION
IF  W !,"vous ne pouvez pas voter, mais vous pouvez fumer"
```

Les actions produites seront les suivantes, sachant que toutes les conditions énoncées sont verifiées :

- passage du curseur à la ligne

- écriture du message :   puisque vous avez 16 ans

- passage à la ligne suivante

- affichage de :   et que vous habitez en Alsace

- passage à la ligne

- affichage de :   vous ne pouvez pas voter, mais vous pouvez fumer

Sans le savoir nous venons d'utiliser la variable système $TEST. Il est à remarquer, de suite, que la forme d'écriture des trois derniers IF apparaît sans qu'une condition soit spécifiée. Ils sont séparés du verbe suivant par au moins deux espaces.

B) La variable $TEST (ou $T)

Cette variable est alimentée par le systeme, au même titre que $X et $Y. Elle contient toujours zéro ou un. En fait, Il s'agit du resultat de la derniere condition évaluée (nous vous rappelons que 0 et 1 sont des contenus logiques faux ou vrais). La variable $TEST est donc une variable logique. Nous radotons certainement, mais on s'aperçoit que les concepts de MUMPS sont définis pour être au plus proche de la forme d'expression humaine.

C) La commande ELSE

De temps en temps, il est nécessaire, dans le cas d'une condition non vérifiéeet seulement dans ce cas, d'exécuter un certain nombre d'actions. C'est ce que permet de faire la commande ELSE. Reprenons le dernier exemple et ajoutons une nouvelle ligne, incluant la commande ELSE :

Attention !!! La commande ELSE est toujours suivie de deux espaces.

```
IF (AGE>15)&(AGE<22)&(REGION="Alsace") W !
IF  W "puisque vous avez ",AGE," ans"
IF  W !,"et que vous habitez en ",REGION
IF  W !,"vous ne votez pas, mais vous pouvez fumer"
ELSE  W !,"vous avez tous les droits"
```

→

Si AGE n'est pas compris entre 15 et 22 ou que le contenu de REGION soit différent de "Alsace" le message vous avez tous les droits sera affiche à la ligne.

RESUME

Dans les deux chapitres précedents, nous avons étudié le concept de la ligne MUMPS en identifiant ses principaux composants et en montrant que des actions multiples peuvent être exécutées sur une ligne. D'autre part, nous avons vu l'exécution conditionnelle d'une ligne entière, ou en partie, à l'aide des commandes IF et ELSE. Rappelons que ces commandes ne s'appliquent qu'à la ligne dans laquelle elles apparaissent.

SUITE DE LA PAGE 2

FR- fr1 fr2 --- fr1-fr2

Effectue la différence de deux fractions. Idem FR*.

FR@ adr --- nn nd

Dépose sur la pile le numérateur nn et le dénominateur nd d'une variable fractionnaire.

FR! nn nd adr ---

Affecte les valeurs nn et nd à la variable fractionnaire dont l'adresse est au sommet de la pile.

FRCONST nn nd --- <mot> en compilation
<mot> dépose nn nd en exécution

Définit une constante fractionnaire.

FRVAR nn nd --- <mot> en compilation
<mot> dépose adr en exécution

définit une variable fractionnaire.

Exemples:

186 1000 FRCONST TVA
(soit 18,60 %)
145 1 TVA FR*
(soit (145,00*18,60)/100))
ce qui donne 26970 1000
(soit 26970/1000 = 26,97)

Pour éviter un dépassement de capacité, on peut procéder comme suit:
145 1 TVA REDUIT FR*
ce qui donne 13485 500
si on réduit 13485 500 REDUIT
on obtient 2697 100

Un conseil quand même, évitez de manipuler des valeurs trop importantes. En général, ces routines seront d'une aide précieuse pour tout lycéen pratiquant l'arithmétique et manipulant des fractions et obligé de ne pas donner de résultat décimal.

```
SCR # 71
( INVERSE, REDUIT, FR*, FR/ )
 : FRINV ( n1 n2 --- n2 n1 )
SWAP ;
 : REDUIT ( n d --- nr dr )
2DUP PGCD
DUP ROT SWAP
/ >R / R> SWAP ;
 : FR* ( fr1 fr2 --- fr1*fr2)
SWAP >R *
SWAP R> *
SWAP REDUIT ;
 : FR/ ( fr1 fr2 --- fr1/fr2)
FRINV FR* ;                                   ;S
```

```
SCR # 72
( FR+, FR-, FRMINUS, FR@ )
 : FR+ ( fr1 fr2 --- fr1+fr2)
>R OVER *
ROT R * +
SWAP R> *
REDUIT ;
 : FRMINUS ( Fr --- -Fr )
SWAP MINUS SWAP ;
 : FR- ( fr1 fr2 --- fr1-fr2 )
FRMINUS FR+ ;
 : FR@ ( ADR --- n d )
2@ ;
;S
```

Tous systèmes

```
SCR # 70
( CALCUL DU PGCD )
 : PGCD ( n1 n2 --- PGCD )
2DUP
< IF SWAP THEN
BEGIN
   SWAP OVER MOD DUP
0= UNTIL
DROP ;
 : PPCM ( n1 n2 --- PPCM )
2DUP PGCD */ ;
;S
```

```
SCR # 73
( FR!, FRVAR, FRCONST )
 : FR! ( n d adr --- ) 2! ;
 : FRVAR ( n d --- <mot>      en compilation )
 2VARIABLE (     --- adr      en interpretation ) ;
 : FRCONST ( n d --- <mot>   en compilation )
2CONSTANT (    --- n d     en interpretation ) ;
 ;S
```

FORTH ... ACE extensions et communication

par Ludwig RICHTER-ABRAHAM

Nous avons reçu le numéro 2 de "VIERTE DIMENSION", dont nous avons extrait ces quelques routines applicables au JUPITER ACE. Ces routines sont présentées en deux parties, la première contient les définitions manquantes les plus utilisées, la seconde des utilitaires de mise au point.

```
16 BASE !                                ( HEX )
: HEX           10 BASE ! ;  IMMEDIATE
: BIN            2 BASE ! ;  IMMEDIATE
: .D  ( n -- )  DECIMAL . HEX ; ( hex vers dec)
: .H  ( n -- )  HEX . DECIMAL ; ( dec vers hex)
: ' .( <nom> -- )   FIND 2+ ;
: ?  ( adr -- ) @ . ;
: 2DUP ( d1 -- d1 d1 ) OVER OVER ;
: NOT ( tf -- ff )  0=  ;
: -DUP ( n -- n n si n<>0 ) DUP IF DUP THEN ;
: ?DUP ( n -- n n si n<>0      -DUP ;
: LATEST  ( -- adr )  CURRENT @ @ ;
: TRAVERSE  ( adr1 n1 -- adr2 ) SWAP
    BEGIN OVER + 7F OVER C@ < UNTIL
    SWAP DROP ;
: PFA  ( nfa -- pfa ) 1 TRAVERSE 5 + ;
: NFA  ( pfa -- nfa ) 5 - -1 TRAVERSE ;
: CFA  ( pfa -- cfa ) 2 - ;
: LFA  ( pfa -- lfa ) 4 - ;
: COUNT  ( adr1 -- adr2 n ) DUP 1+ SWAP C@ ;
: R   ( -- u ) I ;
20 CONSTANT BL
: -TRAILING ( adr1 n1 -- adr1 n2 )
    DUP 0 DO 2DUP + 1 - C@ BL -
            IF  LEAVE
            ELSE 1 -
            THEN
           LOOP ;
: -FIND  ( -- pfa b tf )
    BL WORD HERE CONTEXT @ @ DUP 0=
      IF  DROP HERE LATEST THEN ;
: +-  ( n1 n2 -- n3 )  0< IF  NEGATE  THEN ;
: TOGGLE  ( adr b -- ) OVER C@ XOR SWAP C! ;
: SMUDGE  ( nfa -- ) 20 TOGGLE ; IMMEDIATE
: 0<>  ( n -- flag ) 0= NOT ;
: ?TERMINAL  ( -- flag )  INKEY 0> ;
```

Et voici les définitions spéciales qui ne sont pas décrites par le manuel du Jupiter ACE.

```
HEX
: NOOP   ;
: RAMTOP  ( -- adr ) 3C18 @ ; ( variables sys.)
3C37 CONSTANT DP              ( variables sys.)
3C3B CONSTANT SP              ( variables sys.)
: .S  ( --) CR CR SP @ HERE C + 2DUP -
   IF  ." II TOS "
     DO  CR I @ . 2 +LOOP
   ELSE  DROP DROP
     ." === STACK EMPTY ==="
   THEN CR ;
: HANG  ( -- )  BEGIN ?TERMINAL UNTIL ; ( key?)
: BREAK ( -- ) .S HANG 800 0 DO NOOP LOOP  ;
                   ( point d'arrêt avec attente)
0 VARIABLE SOURCE   0 VARIABLE DESTIN
: CMOVE  ( adrorig adrdest nombre -- )
   ROT ROT 1- DESTIN ! 1- SOURCE !
   0 DO SOURCE @ 1+ DUP SOURCE ! C@
        DESTIN @ 1+ DUP DESTIN ! C!
   LOOP ;
: FILL  ( adr quan b -- ) ROT ROT OVER + SWAP
   DO  DUP I C! LOOP DROP ;
: ERASE ( adr count -- )  0 FILL ;
: BLANKS ( adr count -- ) 20 FILL ;
( utilitaires de DUMP )
: (XCH)  ( c -- bl )  DROP 20 ;
: (TEST) DUP F > IF
   ELSE DUP DUP 18 < IF (XCH) THEN
   THEN
   DUP 8F >
   IF
   ELSE DUP 98 < IF (XCH) THEN
   THEN ;
: (ASC)  DUP DUP 8 + SWAP
   DO I C@ (TEST) EMIT 2 SPACES LOOP ;
: (ZE)   DUP DUP 8 + SWAP
    DO  I C@  DUP 10 <
     IF  SPACE THEN .
    LOOP  ;
: (IND)  6 SPACES 8 0
    DO I SPACE . LOOP CR CR ;
: (HDUMP1) DUP U. SPACE (ZE) CR 8 + ;
: (HDUMP2)  DUP U. SPACE (ZE) CR
    8 SPACES (ASC) 8 + ;
: DU1  CR (IND) 0
    DO  (HDUMP1)  LOOP CR ;
: DU2  CR (IND) 0
    DO (HDUMP2) LOOP CR ;
: ID.  PAD  20 5F FILL
    DUP PFA LFA OVER - PAD SWAP CMOVE
PAD COUNT IF AND TYPE SPACE ;
: SECONDS ( n -- ) 0 DO
    8400 0 DO  LOOP
    LOOP  ;
: MILLISEC  ( n -- ) 10 / 0 DO
    80 0 DO  LOOP
    LOOP  ;
```

par J. HIRSZOWSKI

Le Jupiter ne comportant pas d'interfaces standards, le branchements des périphériques (imprimante, modem, etc...) nécessite à priori une carte d'adaptation au bus.

Pour une liaison série on peut laisser libre le bus en utilisant MIC et EAR; voici un programme pour sortir sur MIC les caractères présentés à l'écran; on obtient une sortie TTL tamponnée, en prélevant le signal en amont du condensateur de sortie (cf. figure 1) pour l'inverser par un simple transistor T; la polarisation de T peut être assurée par le périphérique lui-même, si celui-ci présente une résistance de rappel au +5V, comme c'est le cas par exemple pour le MINITEL. Le format série choisi est: 7 bits par caractère, 1 bit de parité paire et un bit de stop, ceci en 300 bauds.

```
: 7BITS
    7 0 DO  2 /MOD  LOOP DROP ;
: BITPARITE  0  9 2
    DO  I PICK +  LOOP  1 AND  ;
: FORMAT
    7BITS  BITPARITE 0 10 2 DO I ROLL 0=
 ( retournement de la pile on complémente )
 ( les bits sur D3 )
    8 *  LOOP 8   ( bit de start)
    ;
: TBIT  ( bit -- transmission )
    254 OUT  23 0
    DO LOOP  ( délai=duree 1 bit)  ;
: TX  ( car -- emission sur MIC)
    FAST FORMAT
    10 0 DO  TBIT  LOOP SLOW ;
 ( les car sont lus dans la mémoire d'écran)
 ( ainsi, pour recopier la ligne n ...)
: LIGNE  ( no de ligne -- )
32 * 9126 + DUP
32 + SWAP
    DO  I C@ TX  LOOP
13 TX  10 TX  ( CR+LF)  ;
 ( et pour recopier l'écran ...)
: ECRAN
    23 0 DO  I LIGNE  LOOP ;
```

CONTACT

Ludwig RICHTER-ABRAHAM Lilienthalstraße 1
6103 GRIESHEIM RFA   pour les extensions sur Jupiter ACE.

J. HIRSZOWSKI au Club Paris Micro (APAGECISE) 10, rue Erard 75012 PARIS France pour les routines de transmission série.

→

1) Dans le JUPITER:

R=10kΩ C=47nF 2 1

Prélever le signal de sortie sur 1.

EAR MIC

74LS00 74LS74 74LS08 74LS86 74LS02

1 R 2

T=2N2222 Rx Masse

FORTH *TRUCS ET TOURS DE MAIN* par l'ami KRO

HECTOR_HRX : FORTH

UN TONE SANS VIBRATO par MHC
Le mot TONE tel que defini dans le HRX produit un son vibrato. La raison tient au fait que les interruptions du micro sont validees pendant l'emission du son. On peut y remedier en les interdisant durant la production de la note.
Il nous faut d'abord definir les mots DI et EI qui interdiront et valideront les interruptions.

Deux definitions vous sont proposees selon que vous disposez ou non de l'assembleur-Forth pour HRX.

Avec Assembleur Forth :

```
HEX CODE DI DI, NEXT
    CODE EI EI, NEXT DECIMAL
: TTONE DI TONE EI ;
```

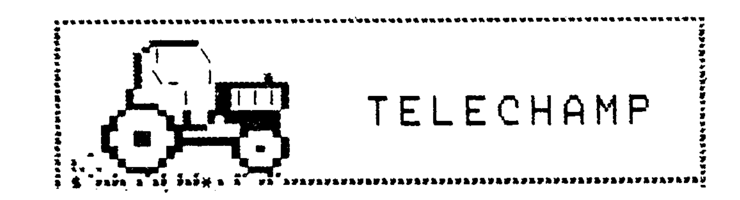
Sans l'assembleur Forth

```
HEX
CREATE DI F3 C, DD C, E9 C, SMUDGE
CREATE EI FB C, DD C, E9 C, SMUDGE
DECIMAL
: TTONE DI TONE EI ;
```

```
( FORTH : DUMP EN HEXA ET EN ASCII )
    ( HECTOR HRX - PAR MHC )

HEX
: 10< C@ DUP 10 <
    IF ." 0" . ELSE . THEN ;
: ?ASC C@ 7F AND DUP 20 <
    IF DROP 20 EMIT ELSE EMIT THEN ;
: DASC SPACE 4 -
    4 0 DO DUP ?ASC 1+ LOOP ;
: !DUMP 4 0
    DO DUP 10< 1+ LOOP DASC ;
: DUMP CR BEGIN ?TERMINAL
    IF DROP QUIT THEN
    CR DUP U. !DUMP AGAIN ;
DECIMAL
```

Relevé sur SAM
TELETEL 3



COMMENT CREER SOUS B3X DES IMAGES FORTH DE GRANDE DIMENSION

Pour pouvoir créer une image FORTH dont la définition dépasse 255 caractères, il suffit de prendre autant d'écrans FORTH que nécessaires et les sauvegarder sur cassette.

Il faut tout d'abord initialiser de la manière habituelle (en mode direct):

```
CLEAR 255,&9FFF
COLD &A000,&B88C
```

```
tapez FORTH QUIT      (vous êtes sous FORTH)
      1 OPEN          (ouvre l'écran n°1)
      EDITOR          (passe sous mini-éditeur
                      avec les commandes habi-
                      tuelles)
      0 P 4 BASE !    (description du dessin)
      1 P 1111 1111 1111
      2 P 2222 2222 2222
      etc...
```

Si le dessin dépasse un écran, ne pas oublier de chainer (-->) et ouvrir le second écran.

Si le second écran ne suffit pas, même procédure, ouvrir le 3ème écran, HECTOR demande alors "ECRAN 1 sur cassette ?" (prévoir une cassette vierge, bande amorce avancée).

Avoir terminé la description du dessin, effectuer la sauvegarde sur cassette par UPDATE

Eteindre HECTOR, puis rallumer et charger B3X

Initialiser CLEAR et COLD.

```
FORTH QUIT
1 LOAD          (va chercher sur la cassette
                l'écran et le compile)
```

revenir au B3X en tapant n'importe quelle touche suivie du RETURN, et enfin taper le programme d'animation du dessin.

MATRICE A TROIS DIMENSIONS en FORTH

Dx - Dy - Dz contiennent les dimensions du tableau.

SURTOUT sert de démonstration pour remplir et vider le tableau avec un nombre croissant de 0 à 999.

Exemple: >TBL SURTOUT remplit le tableau
TBL> SURTOUT affiche le tableau.

ACTION lit ou écrit dans le tableau en fonction de "ACTE" manipulé par >TBL et TBL>. Action n'est utilisé que pour la démonstration de SURTOUT.

Pour écrire des valeurs dans le tableau:

valeur x y z PLACER ou
x y z VALEUR STOCKER

Pour lire une valeur dans le tableau:

x y z LIRE ou
x y z VIDER

Exemples:

```
10 5 5 5 PLACER  met la valeur 10 en case 5 5 5
20 6 6 6 PLACER  met la valeur 20 en case 6 6 6
6 6 6 LIRE       lit la valeur de la case 6 6 6
        et la met sur la pile de données.
5 5 5 LIRE + .   lit la valeur de la case 5 5 5
        la met sur la pile, additionne avec la
        valeur de la case 666 et affiche le ré-
        sultat (30).
30 7 7 7 PLACER  met la valeur 30 en case 7 7 7
7 7 7 VIDER      affiche valeur de case 7 7 7
```

```
ECRAN No   15
 25 MSG" INDICE INVALIDE"
FORGET ME          : ME ;
10 CONSTANT Dx     10 CONSTANT Dy
10 CONSTANT Dz      0 VARIABLE acte
0 VARIABLE cpt
: DIM Dx Dy * Dz * 2* ;
0 VARIABLE TABLEAU DIM 2- ALLOT
: NON 0 SWAP ! ;
: OUI 1 SWAP ! ;
: RAZTBL TABLEAU DIM ERASE ;
: INCLUS?
  OVER > SWAP 1+ 0> AND ;
: PERMIS?          ( a b c -- a b c)
  >R >R
    DUP Dx INCLUS?
  R> DUP Dy INCLUS? ROT AND
      R> DUP Dz INCLUS? ROT AND
  0= IF 25 ?ERROR THEN ;
: ADR+ PERMIS?    ( a b c -- adr)
  Dx Dy * * 2* SWAP Dx * 2* +
  SWAP 2* + ;                       -->
ECRAN No   16
: PLACER           ( n a b c --)
  ADR+ TABLEAU + ! ;
: LIRE             ( a b c -- n)
  ADR+ TABLEAU + @ ;
: STOCKER          ( x y z --)
  4 ROLL 4 ROLL 4 ROLL PLACER ;
: VIDER            ( x y z --)
  CR 3 PICK . OVER . DUP .
  LIRE . ?TERMINAL DROP ;
: >TBL ACTE OUI ;
: TBL> ACTE NON ;
: ACTION
  acte @ 0= IF VIDER
  ELSE cpt @ STOCKER 1 cpt +!
  THEN ;                            -->

ECRAN No   17
: SURTOUT CPT NON
  Dz 0
  DO I
    Dy 0
    DO I
      Dx 0
      DO I
        OVER 4 PICK ACTION
      LOOP DROP
    LOOP DROP
  LOOP ;
```

HECTOR HRX

1re Version

```
       A> PIP  B: ED.COM  =  A:  ED.COM   [R] return
si PIP est sur  Destination   origine
     A
              Nom et suffixe   Nom et suffixe
               Destination      d'origine
```

2 ème version

```
     A> PIP    [R]
```

production de

```
     * [ ]——curseur
```

Vous pouvez changer de disquette en A et donner l'ordre d'échange après le * [ ] mais gare à vous si la nouvelle disquette A n'est pas initialisée CPM. Vous quittez par [Return] ou [Reset] en ayant subrepticement remis une disquette CPM en A.
Sinon gare !!! Finalement je préfère la 1re version avec PIP sur A ou sur B.

3ème Version importante :
C'est l'utilisation de PIP pour charger les terminaux :

Ecran de la console	CON:	ou	CRT:
Imprimante	LST:		LPT:
Perforateur	PUN:		PTP:

a) pour l'obtention par exemple du listing imprimante de SQUARO.COB que vous avez sur COBOL MASTER n° 2 vous mettez en A: le master CPM
en B: le master cobol n° 2 et:

```
     A > PIP      LPT:= B:SQUARO.COB
               attention aux  ":"
```

LPT: peut être remplacé par LST:, à vous d'essayer.
N'oubliez pas les 2 points sans cela, l'imprimante restera muette et le PIP ne reconnaissant pas l'imprimante vous obéira à sa façon en copiant tout simplement SQUARO:COB sur le disque A sous le nom idiot de LPT ou LST !!

b) Il semble d'après la mauvaise notice CPM que les désignations CON: LST: PUN: soient les désignations symboliques CPM et que les désignations CRT: LPT: PTP: soient celles fournies en équivalence par le IOCS (Impert Out put Control System) donc modifiables par l'usager en fonction de ses besoins et de ses périphériques.

(programme à écrire en genre d'ASM assembleur)
A moins que ce ne soit exactement le contraire !!!
La suite au prochain numéro.

10) ED.COM Pour l'écriture, lecture du programme source ; n'admet qu'une version ;

```
     A > ED  XXX.SUF  [R] return
ou   A > ED  B: XXX SUF est en B
```

Si XXX.SUF n'existe pas on vous dit NEWFILE et 1 * [ ] curseur et vous n'avez plus qu'à travailler comme nous le verrons plus loin. Vous pouvez faire * Q [R] pour quitter on vous demande Y/N ? Yes ou No(étonné que vous ne vouliez sauvegarder votre chef d'oeuvre !)Sinon vous auriez fait E pour sauvegarder.

11) Des instructions qui n'appellent pas de programme : REN et ERA. Elles sont disponibles dans toute disquette initialisée CPM.

```
     ERA          A: XXX.SUF    [R]
    Erase     Drive  Nom  Suffixe
```

Ecrase le nom du programme dans la Directory (index du disque).
Attention cela marche sans prévenir et vite fait !

```
     A> REN        B:YYY.suf 2  =  XXX.SUF
disque où  Rename se trouve
```

Sur le disque B: vous changez le nom du fichier XXX.SUF en le nouveau nom YYY.suf 2

D. PREPARATION DES 2 DISQUETTES

1) Début | Formater 2 disquettes neuves d 1 et d 2.
Voir C 2
Initialiser et étendre à 56 K — Voir C 4 par COPY...../S

2) Vous êtes prêt: Prenez CPM Master en A et successivement d 1 et d 2 en B.
Vous allez transporter PIP.COM depuis CPM MASTER jusque sur d 1 puis sur d 2. par

```
     A> PIP      B:PIP.COM = A: PIP.COM
Vous êtes en A  Destination    Origine
  avec CPM MASTER
```

autrement dit PIP se transporte lui-même à l'aide de lui-même depuis A: sur B: bel exploit indispensable.

Répétez sur d2.

3) Faites pareil pour ED.COM. Votre éditeur de texte indispensable successivement en D 1 et D 2.

   A PIP B:ED.COM = A:ED.COM

   Vous êtes prêt à affronter COBOL.

4) Avec d 1 en A et COBOL (1) Master en B vous copiez sur d 1 en A.

	COBOL.COM	29 K	
puis	COBOL 1.OVR	12	OVR veut dire Overlay
	COBOL 2.OVR	13	
	COBOL 3.OVR	18	
	COBOL 4.OVR	7	
		79 K	

79 K au total.

Ces 4 programmes serviront en une fois à la compilation de votre programme ESSAI.COB (ESSAI = Votre nom ; COB = source Cobol) qui deviendra Essai.PRN et Essai.REL. Essai.PRN étant votre source sans guère de modif si pas de faute ou les fautes soulignées et Essai.REL le résultat compilé qui serait bon à "chaîner" s'il n'y a pas d'erreurs signalées dans Essais.PRN.

Cobol appelle lui-même Cobol 1, 2, 3, 4.

Votre d 1 est prête, nous l'apellerons COBOL:En tout avec PIP (8K) et ED (7K): 94 K sur 126 que vous avait laissé le CPM systèm sur d 1.

Ordres nécessaires :

```
A> PIP  A: COBOL.COM   = B:COBOL.COM   [R]
A> PIP  A: COBOL 1.OVR = B:COBOL1.OVR  [R]
A> PIP  A: COBOL 2.OVR = B:COBOL2.OVR  [R]
A> PIP  A: COBOL 3.OVR = B:COBOL3.OVR  [R]
A> PIP  A: COBOL 4.OVR = B:COBOL4.OVR  [R]
```

C'est long, mais éfficace.

5) A votre d 2 maintenant (elle est pourvue de (CPM, "56K", PIP, ED,).

   a) avec d 2 en A et COBOL MASTER 1 en B copiez COPLIB.REL sur d 2. C'est la librairie de routines sous programmes dont certains seront linkés (chaînés) à votre propre programme (Exécution longue car COPLIB fait 35K à lui tout seul).

   Ordre A> PIP A: COPLIB.REL =B: COPLIB:REL [R]

   b) d 2 en A: Cobol Master 2 en B: cette fois vous copiez L 80.COM et CRTDRV.REL L 80 sera le programme de Linkage qui ajoutera à votre 1re compilation XXX.REL, les routines nécessaires puisées soit dans COBLIB. soit dans CRTDRV (pour le clavier écran) des verbes ACCEPT et DISPLAY (entrée clavier sortie écran dans Cobol) et d'autres ordres pourront mettre ces routines en action.

ORDRES :

```
A> PIP  A: L 80.COM   = B:L 80.COM    [R]return
A> PIP  A: CRTDRV.REL = B/CRTDRV.REL  [R]
```

Vérifiez par DIR et STAT que vous avez bien les programmes demandés.

Si cela s'est mal passé plusieurs fois sur la même disquette, il vaudrait mieux tout recommencer pour cette disquette.

Nous appellerons d2 ⟶ LINK. (en souvenir de L 80 Programme maitre de cette disquette).

PRECAUTION : Prendre copie de suite de d1 et d2... (Très rapide car vous copierez toute la disquette comme en C 3.)

Nous avons maintenant 2 disquettes Master Personnel qui peuvent nous servir à écrire nos programmes sources (principalement sur LINK ou nous avons ED et plus de 60 K de libre. Nous y obtiendrons successivement XXX.COB après avoir écrit la source, XXX.PRN, et XXX.REL après compilation éventuellement des XXX.$$$ si certaines manoeuvres ont été ratées.

Eventuellement, nous aurons besoin d'autres linkages. Mais pour le moment des petits programmes conversationnels pourront tourner. ; et enfin notre fameux XXX.COM programme fini et bon à éxécuter. Après, tout fini avec bonne satisfaction, et au besoin avoir recommencé 2 fois ou plus !, on nettoie avec ERA et on ne conserve que XX.COB et XXX.COM. Eventuellement on transporte les programmes XXX.COM sur une autre disquette pour garder "COBOL" et "LINK" bien propre. C'est également la technique que nous avons adopté pour "Pascal". Mais ceci est une autre histoire.

E. FAIRE SON PROGRAMME.SOURCE

Le programme-source doit être écrit en ASCII. Une première idée serait de l'écrire comme si c'était un programme en condensé, mais par Save "XXX", A, qui sauve en ASCII et de le retrouver, non par LOAD, mais par MERGE. Mais, il est plus simple et plus certain (pour moi) de le créer sans l'aide de M BASIC en restant sous CPM ⟶ A> à l'aide de la fonction ED.

1) 1re approche Votre disquette link en A, et Master Cobol n°2 en B copiez sur link (par PIP) le SQUARO.COB qui vous est fourni à titre d'exemple.
Faites A>ED (A:) SQUARO.COB
(Facultatif puisque vous êtes en A> ·)

Il vient
1 ◡ ◡ * □
Curseur

Vous êtes à la 1re ligne du texte SQUARO et vous pouvez penser pouvoir lire, deleter, ou modifier. Il n'en est rien, vous pourrez seulement écrire en faisant 1 ◡ ◡ *I [R] return
ce qui donne 1 □→ curseur
et vous écririez.

En réalité le texte n'est pas en mémoire, et ED n'a fait que vérifier si le fichier texte existait. S'il n'avait pas existé sur cette disquette, il vous aurait dit :

NEW FILE
1 ◡ ◡ * □ Curseur

a) Commande *n A
Pour avoir Squaro en mémoire et le lire par ED il faut faire :
1 ◡ ◡ *200 A [R] return
Vous faite 200 pour avoir les 200 lres lignes en mémoire. C'est bien plus que la longueur de SQUARO qui n'en fait guère que 60 environ. Vous auriez pu faire 1 * 70 A [R] ou ⊥ *#A
Après lecture du disk, vous êtes remis à 1 ◡ ◡ * □

b) Commande *Bnt Maintenant faites, 1 * B 10T [R] et vous lisez à l'écran les 10 lres lignes ou vous faites 1 * B#T [R] Vous lisez tout et on vous replace en ⟶ 1 ◡ ◡ * □

c) Commande n1 ◡ ◡ * [R] return
Affiche la ligne suivante (n1 + 1) et vous replace au début de la ligne n° 1. (n1 étant le n° de la ligne où vous vous étiez placé dans le texte.

n1 + 1 [Texte]
n1 + 1 * □

Vous pouvez donc défiler tout votre texte ligne à ligne.

d) Commande n1 ◡ ◡ *ℓ (+ n'est pas accepté)
ou n1 ◡ ◡ *-ℓ

ℓ est le nombre de lignes de déplacement en + ou en -.
Affiche n 2 [Texte]
n 2 * □ curseur
avec n 2 = n 1 + ℓ ou n 2 = n 1 - ℓ et vous replace au début de n 2.

e) Commande n 1 ␣␣ * ℓ K........ ATTENTION.
C'est la commande KILL (tue), délete, détruit ℓ ligne, y compris n 1, et vous replace au début de la 1re ligne que vous venez de détruire :

n 1 * □

Quant à moi, il vaut mieux, après le Kill, faire tout de suite une acceptation (validation) en faisant CTlez [R] appuyer en même temps sur CTLE et Z. (CTLE Z" sera à faire aussi après la fonction (Insert). Après "CTLE.Z" vous êtes remis sur la même ligne n 1 * □.

Pour déleter 1 ligne il faut faire *1 K et non pas *K. Les lignes suivant le Kill sont non seulement détruites mais tous les Nos de lignes conservés diminuent de ℓ et viennent se placer à la suite de n 1.

f) enfin Commande n1␣␣*I [R] ⟶ (Insert)
C'est la commande qui vous permet d'écrire et vous livre la ligne :

n 1␣␣ □ curseur

Vous n'avez qu'à écrire, finir par [R]. Vous êtes replacé en :

n 1 + 1␣␣ □

Vous pouvez écrire votre 2ème ligne et [R] etc... Ou bien, à ce moment, vous acceptez par CTLE.Z. ([CTL] et [Z] ensemble) et ED vous replace :

n 1 + 1␣␣ * □

Pour les 1res fois vous pouvez vérifiez en faisant

n 1 + 1␣␣ * - 1 [R] et :
n 1 [votre texte]
n 1 * □

g) fin
Quand vous avez fini vous faites :

1) Mise sur disque (attention automatiquement sous l'ancien nom donné au moment de ED XXX avec écrasage de l'ancien XXX s'il existait) donc vous faites :

n 1 * E [R]

et après mise sur disque, vous êtes remis en A> ou B>.

2) Ou bien si vous désirez abandonner la nouvelle version

n 1 * Q [R] (pour Quit)

et par prudence ED vous fait confirmer

QUIT (Y/N) □ curseur

Si vous répondez Y, vous êtes remis en

A> ou B> si vous étiez en B>.

h) Je ne conseille pas (les autres commandes).

± ℓC, ± n D, ± N T, F, qui ne m'ont apporté

que des heures et des heures de déboires.
Pour les modif, à la suite de vos erreurs, vous avez fait Kill (par *1 K [R])
Et vous avez :

n 1 * 1 K □
n 1 * □ faites un Insert * I

et retapez la ligne modifiée, [R], et [CTLE Z]
d'où

n 1 * I
n 1 [Texte] [R]
Contrôle Z
n 1 + 1 * □

Vous vérifiez par * - 1 [R]
et vous continuez...

i) Pour pouvoir vous admirer, vous pouvez imprimer le texte (après l'avoir sauvé par * E) par :

A> PIP LST: =A:XXX

(avec imprimante allumée !).
ou PIP LPT: Essayez les 2 et surtout n'oubliez pas les ":".

j) Je ne vous ai pas rappelé les règles d'écritures Cobol.
Ecriture à partir du 8e jusqu'au 11ème (par prudence du 9ème) pour les DIVISION, les SECTION, les niveaux 01, Les paragraphes de la procédure.

Ne rien mettre après le fatidique "."
Ne pas mettre de blanc avant le "."

Pour faire "Remark" ⟶ Tapez un "*"
exactement en 7ème position c.a.à. après espacement de 6.

```
  1 2 3 4 5 6 7
* □ 1 2 3 4 5 6
  └──────────────→
```

L'espacement pour écrire à partir de 9 peut s'obtenir en tapant sur [→] puis [Espace] et après le 9 Tapez sur espace. Retour arrière par [←] (avec effacement).

i) enfin : Bonne chance et Bon courage.

Il vaudrait mieux avoir maintenant appris (un peu!) le Cobol.

Attention : après un Accept D 1 avec D 1 PIC 9 (7). Les chiffres négatifs vous seront refusés, il faut S 9 (7).

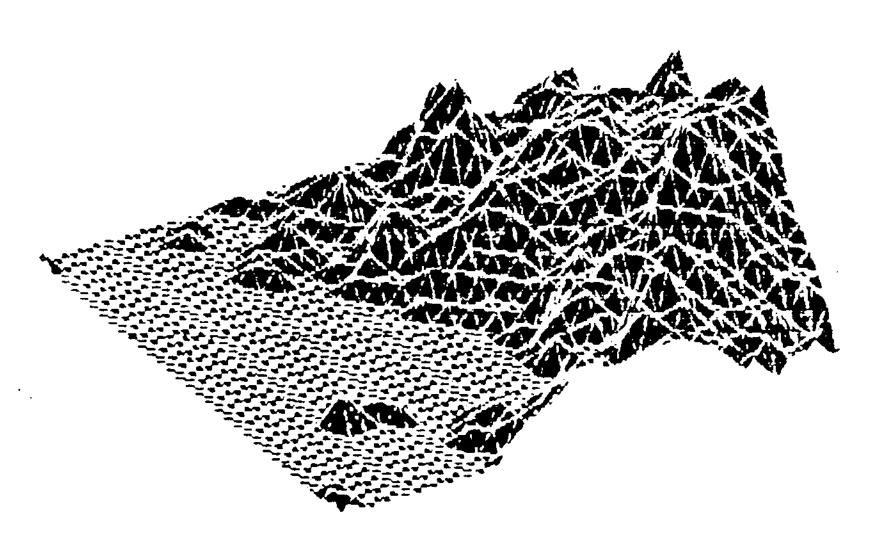
C'est personnellement la seule erreur que j'avais fait dans mon 1er petit COBOL Source.

Montagnes Fractales

en Basic ...

... sur BBC.

Le Basic BBC permet la définition de procédures récursives avec variables locales.



FUTURSYS

FUTURSYS est un micro-ordinateur portable dédié à l'intelligence artificielle, ce qui ne correspond à rien tant que l'on n'a pas défini ce que l'on entend par le terme "intelligence artificielle". A notre sens - et il y en a de nombreux autres - cette discipline consiste à reconstruire les capacités d'adaptation et de création humaines, celle-ci étant pour nous les meilleurs critères d'intelligence.

Sur ce critère, il parait difficile de construire des machines supérieures à l'homme, puisqu'elles seront toujours, elles et les autre machines qu'elles pourront engendrer, des produits de son intelligence.

Ceci étant, si l'on fait aussi bien, on peut faire mieux et ces machines pourraient être supérieures à l'homme au niveau de leurs performances: vitesse, capacité de travail, et la possibilité de les construire en série serait intéressante pour les mêmes raisons, si tant est que l'éducation, ou la sensibilité propre, de ces machines puisse leur être "apprise" rapidement.

Si aujourd'hui nous sommes loin d'en être arrivés à ces problèmes, nous allons cependant essayer de montrer en quoi FUTURSYS est moins éloigné de cette intelligence que des systèmes plus classiques basés sur FORTRAN, COBOL, BASIC etc...

Une première objection à ce que nous voulons démontrer serait de dire que FUTURSYS, comme comme tous les autres ordinateurs, fonctionne avec des suites d'instructions de langage machine; proposition certes vraie, mais objection non fondée, car deux organisations différentes fondées sur le même assemblage de base (molécules) produisent des résultats fort différents à priori (homme et insecte, pour exemple).

Nous ne dirons pas que des structures d'ordinateurs autorisant un traitement plus parrallèle qu'aujourd'hui, seraient inintéressantes. Il n'en demeure pas moins qu'au prix d'une perte de rapidité certaine, un monoprocesseur pourra effectuer les mêmes tâches qu'un multiprocesseur.

L'objet de FUTURSYS a donc été, en tenant compte de cette restriction, d'élaborer une architecture, certes perfectible, mais cependant plus à même de rendre compte de certains phénomènes relevant, je pense, à ce que nous pensons de l'intelligence.

LES PERCEPTS DE FUTURSYS

En premier lieu, il semblerait qu'une caractéristique de l'intelligence soit de pouvoir retrouver rapidement, dans un cerveau, l'ensemble, ou une partie de celui-ci, des images relatives et donc peut-être intéressantes, à l'état dans lequel se trouve le cerveau à cet instant.

De façon plus précise, cet état du cerveau pourrait être défini comme l'état électro-physico-chimique du cerceau, conséquence actuelle des percepts (signaux reçus de l'extérieur) et de l'activité interne passés et actuels.

Nous employons le terme image non au sens d'une vision, mais dans celui d'un ensemble d'éléments.

En ce qui concerne le mot relatif, nous dirons que les images relatives à l'état du cerveau sont celles qui possèdent des éléments activés dans ledit état du cerveau. Nous essaierons de définir le mot activité en caractérisant les éléments activés par le fait qu'ils ne sont pas dans leur état moyen ou qu'ils en changent. Enfin, un élément sera un neurone, ou un ensemble de neurones.

PARALLELISME FUTURSYS ET CERVEAU

Nous allons maintenant dresser un parrallélisme relatif entre un cerveau et FUTURSYS.

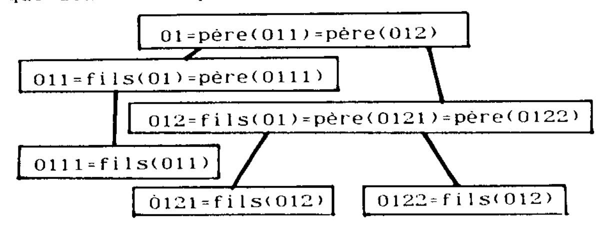
Les percepts de FUTURSYS sont les interruptions et les messages qu'il peut lire sur l'écran. FUTURSYS possède également quelques neurones moteurs lui permettant d'écrire sur l'écran.

Les images de FUTURSYS sont des arbres appelés faits. L'état de FUTURSYS est un arbre représentant "l'expression à évaluer", plus l'ensemble des images ainsi que quelques registres internes pouvant modifier son comportement.

Les images de FUTURSYS relatives à son état sont celles qui peuvent, sous certaines conditions, s'identifier avec l'arbre, ou une partie de celui-ci, représentant "l'expression à évaluer".

On peut faire plusieurs remarques:

- tout d'abord, nous avons introduit le terme d'arbre: un arbre est un ensemble d'objets organisés de la façon suivante: un objet quelconque de l'arbre a un père et Ø ou 1 ou plusieurs fils, pères et fils étant eux-mêmes des objets de l'arbre. Un seul objet n'a pas de père. Ce qui donne la représentation suivante:



à noter qu'avec FUTURSYS, les objets ayant au moins un fils sont toujours appelés des structures.

Il apparait que l'organisation interne de FUTURSYS est l'arbre (ou ensemble d'arbres); c'est une restriction nécessitée par des raisons d'efficacité pratique évoquées plus haut.

- les images de FUTURSYS sont des arbres (les faits): il n'existe que deux types de relation entre deux objets, soit père, soit fils soit rine du tout; nous n'avons pas de relations analogiques. Il s'ensuit, d'une part, que les modifications des images se font sous forme de cassure ou de fusion, et d'autre part, que les limites d'une image sont précises.

- les images de FUTURSYS relatives à son état sont celles dont une partie bien précise s'identifie à l'arbre "expression à évaluer", ou à un sous arbre de celui-ci, priorité étant donnée aux sous arbres; ces conventions restrictives sont par contre très intéressnates pour la "compréhensibilité classique des calculs".

Une autre caractéristique de l'intelligence serait de modifier "consciemment" ou non les images du cerveau. FUTURSYS a la possibilité de modifier les siennes "consciemment" (grâce à d'autres images, en utilisant celles-ci sous forme de connaissance explicite), c'est à dire d'une certaine façon, logiquement, mais n'a pas aujourd'hui celle de le faire "inconsciemment". Cette restriction semble d'ailleurs commune à tous les systèmes opérationnels actuels; elle est due à la mauvaise connaissance des mécanismes d'apprentissage" humains, connaissance qui ne peut que s'améliorer.

Après cette introduction sur les rapports huentre l'intelligence et FUTURSYS, nous verrons dans de prochains articles, l'organisation et le fonctionnement de ce micro-ordinateur.

BASES DE DONNEES ET SYSTEMES EXPERTS

par:
Daniel LANG
Philippe RAFORTHO
Mohamed TOUAJ

II INTEGRATION D'UN SGBD AU SEIN D'UN SYSTEME EXPERT

L'application présentée ci-dessous a été élaborée pour permettre à un système expert de posséder une base de connaissance imporatnte. Les informations contenues dans la base de données représentent la base de connaissance du système expert, et sont directement compréhensibles par celui-ci. Voici donc la présentation du SGBD Ensembliste VORAS.

Développé par l'Université Technologique de COMPIEGNE depuis 1975, ce modèle est inspiré de la théorie des FRAMES et des RESEAUX SEMANTIQUES; il part de l'idée de réaliser une décomposition sémantique du réel. Le réel est représenté en termes d'individus et de relations entre ces individus.

Le modèle Ensembliste présente un format standard ed représentation des données:

- E: entités, objets concrets ou abstraits
- P: propriétés
- V: valeurs associées aux propriétés

Tout élément d'information ayant des attributs uniques peut se mettre sous un format d'entité.
Exemple:

navire (modèle) navire.5 (objet de type navire)
propriété.nom nom Clémenceau
propriété.genre genre porte-avions
Personne (modèle)
propriété.nom
propriété.prénom

On distingue deux types de propriétés:

- propriétés terminales: elles correspondent aux attributs de l'entité pour lesquels une valeur "immédiate" est associée à celle-ci. Par exemple, dans l'entité Personne, nom et prénom sont des propriétés terminales.

- propriétés de structure: ce sont des propriétés dont l'information associée est une liste d'entités. Elles assurent le lien entre deux entités.

Exemple:

dans l'antité Navire, propriété terminale: nom, genre
propriété de structure: capitaine
d'où Navire.5 nom Clémenceau, genre porte-avions, capitaine personne.5

Les valeurs (V) associées aux propriétés sont des informations immédiates correspondant aux propriétés terminales d'une entité.

La propriété de structure Capitaine permet d'associer un Navire à une entité différente Personne.5, cette dernière pouvant être impliquée dans d'autres liaisons avec des informations indépendantes de la notion de Navire. On peut supposer que cette même personne, Personne.5, a écrit un livre; une entité Livre.37 est associée à cette personne par une propriété de structure :Auteur.

A chaque propriété de structure, une liaison inverse est créée automatiquement.

Livre Livre.37
(propriété terminale) Titre Titre La marine (à voiles
(propriété de structure) Auteur Auteur (Personne.5

Personne.5
Nom Durand
Prénom Léon
capitaine de Navire.5
Auteur de Livre.37

Les objets Livre.37 et Navire.5 possèdent une une information en commun. Plutôt que de la reproduire dans chaque entité, cette information est synthétisée sous la forme de l'entité Personne.5 stockée de façon unique et indépendante ce qui facilite les modifications.

Imaginons que le Clémenceau change de capitaine, il suffit de supprimer la propriété de structure entre Navire.5 et Personne.5 (propriété de structure: Capitaine et liaison inverse Capitaine de) pour mettre en place le nouveau lien sur la personne du nouveau capitaine.

MODELE ENSEMBLISTE (Méta Modèle)

On définit les propriétés comme des objets utilsables par toute entité qui les possède comme attributs (Nom est une propriété commune à Navire et à Personne). On utilise le format ensembliste pour les modéliser et on y adjoint des propriétés "système" pour assurer leur traitement par le SGBD. Ainsi on construit la propriété Nom comme une entité possédant des attributs fonction de saisie, fonction d'impression, opérateur de filtrage...

L'élaboration de toute propriété terminale est facilitée par l'utilisation du Méta Modèle "entité propriété terminale" regroupant la liste exacte de ces attributs. Ces derniers sont spécifiques à la propriété de structure (comme Capitaine) à l'aide du Méta Modèle "entité propriété de structure".

De façon générale, la création puis la modification en cours d'évolution des modèles sont facilitées par l'existence de Méta Modèles.

ACCES A L'INFORMATION

Le mécanisme d'accès dans la base de données ensemblistes est basé sur la mise en place (conjointement à la saisie des objets décrits) d'un jeu de points d'entrée sur certaines propriétés caractéristiques des entités. Les points d'entrée se définissent lors de la création du modèle de la propriété terminale. Le point d'entrée est une clé d'accès, l'information associée à cette clé est un chaînage inverse. Ce chaînage inverse pointe sur des entités ayant une propriété terminale dont la valeur correspond à ce point d'entrée.

Exemple: (Clémenceau (Nom de Navire.5 Person(ne.25)
(Titre de Livre.35 ...))

Dans cet exemple, Clémenceau est à la fois le nom d'un Navire.5 d'une Personne.25, et aussi le titre d'un Livre.35.

Dans le cas de requêtes simples, rechercher une entité ayant une propriété s'effectue en synthétisant le point d'entrée correspondant à la valeur associée à la propriété. On charge la liste associée, pointant directement sur les entités corrsepondantes (exemple qui a pour nom Clémenceau, on trouve Navire.5 et Personne.25). Pour des requêtes plus complexes, on fait intervenir une combinaison de points d'entrée.

Une deuxième voie concerne le filtrage sur un ensemble d'entités. On effectue un premier accès par points d'entrée, puis on filtre la liste des objets obtenus à l'aide d'opérateurs associés aux propriétés terminales.

exemple: quelle est la personne qui a pour nom Clémenceau ?

LE SGBD VORAS: UNE AIDE AU SYSTEME EXPERT

Les règles de production constituent une représentation satisfaisante, car très modulaire, du savoir-faire. Cependant, plus la base de connaissance est vaste, plus les performances du du système deviennent médiocres.

L'idée principale du couplage d'un système expert avec le SGBD ensembliste VORAS est de gérer l'ensemble des règles de production nécessaires au système expert, par un SGBD.

Chaque production correspond à un objet accessible de la base. Alors que dans la plupart des systèmes experts, les règles sont organisées en groupes associés à des sous-problèmes, dans un environnement Ensembliste il est facile de créer un système de production très important. Ceci se fait sans avoir à partitionner manuellement les règles en classes. Dans le processus de contrôle, la phase d'extraction de la liste minimale des règles nécessaires à l'exécution d'un contexte est automatique. L'automatisation est rendue possible grâce aux possibilités d'accès aux productions sur des critères multiples fournis par les mécanismes propres à ce type de base.

MECANISMES DE CONTROLE

Les systèmes fondés sur l'emploi de règles de production se sont avérés très efficaces. Cependant, lorsque la complexité du domaine visé s'accroît, les règles deviennent élaborées et très nombreuses. Ceci fait apparaître le problème du contrôle au niveau de l'exécution.

Une premièr solution à ce problème, permettant de déduire et de réordonner l'ensemble des règles à chaque cycle par examen de leur contenu, es d'utiliser les méta-règles.

Une autre approche consiste à implanter le processus de contrôle en utilisant une grammaire de règles.

Mais, les deux solutions impliquant l'examen des règles présentes en mémoire centrale à chaque cycle, sont difficilement réalisables dans le cas de base de connaissance très volumineuse.

Cette étude propose une nouvelle approche, s'appuyant sur la structure d'une base de données Ensembliste. Elle permet d'organiser les règles comme des entités dans une base de données et d'utiliser le mécanisme d'interrogation associé à cette base, pour extraire à chaque cycle des sous-ensembles de règles pertinentes dans un contexte dónné. Ainsi, les règles de production sont implicitement groupées en ensembles virtuels. Une structuration sémantique des règles accessibles par les points d'entrée sur les informations qu'elles contiennent est établie. La gestion des associations entre règles, qui est du ressort du SGBD VORAS, est entièrement automatisée et transparente pour l'utilisateur.

MECANISME D'EXTRACTION

Dans ce système, chaque production est un élément de connaissance représenté dans la base de données et indexé sur son contenu (certains mots clés). Le mécanisme d'indexation est piloté par la base. Il utilise principalement le système de création des points d'entrée du SGBD. Toute modification du contenu des règles engendre une mise à jour automatique des listes de production associés aux points d'entrée. L'indexation est effectuée contrairement aux systèmes classiques sur plusieurs critères simultanément. A la saisie, le système classe les règles dans des listes correspondant aux divers mots-clés qu'elles contiennent. A l'exécution, la diversité des possibilités d'accès aux règles est comparable à ce que l'on obtiendrait par l'unification sans impliquer le fastidieux examen de toutes les règles.

STRATEGIE DE SELECTION

Si l'on ne prend pas de précautions, le phénomène de saturation est inévitable dans un système important. L'augmentation des règles engendre un nouveau type de dépendance lié à l'ordre de placement dans la liste des règles à exécuter. Il faut donc un mécanisme de stratégie de sélection. Dans la plupart des systèmes experts, un jeu de règles et un jeu de méta-règles sont retirés de la base. Les méta-règles sont lancées pour traiter la liste des règles associées au même but. Le critère de sélection est au départ unique, c'est la notion de but. Alors que dans le système élaboré à l'U.T.C, ce critère est multiple et la sélection des règles est plus fine. L'ensemble des règles déterminées par la phase de retrait est donc plus restreint, et le sméta-règles ne sont plus nécessaires. En outre, le jeu de méta-règles risque de s'accroître simultanément avec l'augmentation du système réel pour aboutir à une saturation totale, d'où l'inconvénient des méta-règles. Le besion de réordonner la liste des règles exprime le problème peu évidend de leur indépendance. Le fait qu'une règle mérite d'être placée avant une autre parce qu'elle a plus de chances d'être exécutée est une forme de relation entre les règles. La situation devient délicate lorsque le nombre de relations de ce type croit, ce qui va de pair avec l'augmentation de la liste des règles. C'est pourquoi dans le système étudié ici, le nombre de règles est assez réduit pour chaque situation, dans le cas contraire, on subdivise le problème en sous tâches.

MODELISATION DES REGLES DE PRODUCTION

Une règle est une entité (telle qu'elle est définie dans le modèle Ensembliste) possédant en plus de la propriété Nom qui permet de l'identifier, deux propriétés terminales:

" Prédicat " et " Action "

Le but de ce modèle est de définir la liste des attributs des règles et ed guider leur saisie ainsi que leur mise à jour. (voir figure 1)

INDEXATION DES REGLES SUR LEUR CONTENU

Il convient de limiter le nombre de règles à brasser à chaque étape, c'est pourquoi on leur associe des points d'entrée sur les informations qu'elles traitent. Il est alors possible de les atteindre par l'utilisation du mécanisme d'accès à la base de données, et l'on peut extraire automatiquement à chaque changement de contexte le sous-ensemble optimal de règles. La mise en place de ces points d'entrée est effectué automatiquement lors de la saisie de la règle, par une fonction qui examine les parties prédicat et action. Toutes ces opérations sont totalement transparentes pour l'utilisateur.

PARTIE PREDICAT

On crée un point d'entrée sur chaque variable corespondante à un mot-clé de la règle. Il permet de regrouper les règles faisant référence à une même variable. Le principe est le suivant:

Chaque fois que l'on rencontre un mot-clé dans une clause (par exemple CONTACTL), on vérifie s'il existe une entité variable qui porte ce nom dans la base.

- Si oui, la règle (REGLE.37) est liée à cette entité
- Si non, l'entité est créée (VAR.16) et l'opération de liaison est effectuée ensuite.

On dispose alors d'un point d'entrée sur la varaible par son nom (CONTACTL). Cette variable est liée à toutes les règles qui l'utilisent. C'est ainsi que l'entité VAR.16 est accessible dans la base par son nom CONTACTL qui constitue un point d'entrée. Sous cette entité, on retrouve dans le champ Variable Prédicat la liste des règles de production qui se réfèrent à CONTACL en tant que variable prédicat (VARP.DE REGLE.37). Si la règle est modifiée par la suite, et qu'une variable n'est plus utilisée dans une clause, le lien entre la règle et la variable sera suprimé. Cette opération est transparente et s'effectue lors de la mise à jour de la règle.

→

```
Exemple d'une règle :                                   figure 1.

 REGLE.37
        nom:                  règle-contact-courant

        prédicat: (AND
                    (NOT (NULL CONTACTL))
                    (LTCARD CONTACTL 4)
                    (NOT (NULL CONTACT)))

        actions:  (SETL ORIGINE (+(GETM ORIGINE)(GETM PAS)))
                  (MOVEY CONTACT (LIST (GETM ORIGINE) 0 8))
                  (SETL CONTACTL (CONS (GETM CONTACT)
                                        (GETM CONTACTL)))

        variables prédicat (noté varp):
                  VAR.16 VAR.21

        variables actions (noté vara):
                  VARD.15 VARD.32 VAR.21 VAR.16

 VARD.15                             VARD.32
    nom:       ORIGINE                  nom:      CONTACT
    VARA.DE:   REGLE.37                 VARA.DE:  REGLE.37

 VAR.21                              VAR.16
    nom:       CONTACT                  nom:      CONTACTL
    VARA.DE:   REGLE.37                 VARP.DE:  REGLE.37
    VARP.DE:   REGLE.37

 VARD
    nom-externe:               variable destination
    propriétés-terminales: nom              ;point d'entrée

 VAR
    nom-externe:               variable lue
    propriétés-terminales:  nom             ;point d'entrée
```

PARTIE ACTION

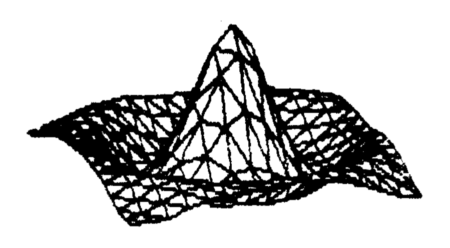
Le système construit un point d'entrée sur la variable destination ainsi que sur chaque variable référencée dans l'expression évaluée. Le traitement est analogue au précédent. Après avoir vérifié s'il existe une entité variable correspondant au mot clé, il y a crétion éventuelle et lien entre l'entité variable et la règle de production. Un point d'entrée existe sur le nom de la variable et permet par analyse de celle-ci de retrouver les règles qui s'y réfèrent. La seule différence réside dans le fait que pour les variables destination, l'entité créée possède un nom interne (VARD) distinct de celui des variables simplement lues en mémoire centrale (VAR), dans ce cas, le nom interne est le même que pour les variables prédicats. Cette distinction permet un affinage du critère de sélection.

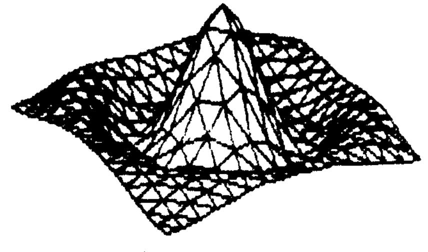
EFFICACITE

Dans les systèmes classiques, soit la totalité des règles est présente en mémoire centrale et l'interpréteur doit brasser la zone des prédicats de nombreuses règles avant de trouver la bonne, soit on a affaire à un méta-système qui améliore les performances dans la mesure où il trie une liste de règles de taille moyenne et figée. Dans le système présenté ici, le processus d'extraction détermine une liste réduite de règles, de plus il n'est pas appelé systématiquement à chaque cycle.

En conclusion, ce système améliore très nettement les performances d'un système expert en lui permettant d'avoir une base de connaissance importante. Néanmoins, pour l'utilisation du point de vue des bases de données, VORAS impose un nouveau formalisme, ce qui interdit l'utilisation du modèle relationnel. Cet inconvénient est amplifié par le fait que ce modèle n'existe pratiquement qu'au sein de l'U.T.C. Nous pouvons de plus poser la question suivante: est-ce que le modèle ENSEMBLISTE est au moins aussi puissant que le modèle relationnel ?

■





réalisé en Basic sur BBC.